COMPARATIVA DE DESEMPEÑO EN LA CARGA DE ESCENAS ENTRE LAS LIBRERÍAS DE GRÁFICOS 3D VULKAN Y OPENGL

MANUEL FERNANDO SABOGAL OCAMPO

UNIVERSIDAD TECNOLOGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PEREIRA

2019

COMPARATIVA DE DESEMPEÑO EN LA CARGA DE ESCENAS ENTRE LAS LIBRERÍAS DE GRÁFICOS 3D VULKAN Y OPENGL

MANUEL FERNANDO SABOGAL OCAMPO

TRABAJO DE GRADO PARA OPTAR POR EL TÍTULO DE INGENIERO DE SISTEMAS Y COMPUTACIÓN

DIRECTOR:

CARLOS ANDRES LOPEZ

UNIVERSIDAD TECNOLOGICA DE PEREIRA

FACULTAD DE INGENIERÍAS

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

PEREIRA

2019

AGRADECIMIENTOS

Primero que todo quiero expresar mi agradecimiento a mis padres, han sido las personas que me han apoyado toda mi vida y han estado ahí para las buenas, las malas y la realización de este proyecto.

A mi hermano por ser un ejemplo a seguir y mi mentor cuando tenía dudas, seguramente sin él no hubiera podido llegar hasta acá.

Al profesor Carlos Andrés Lopez, no solo por haber sido mi director de grado, si no por también haber soportado y haberme apoyado el tiempo que me demoré en realizar este proyecto.

Finalmente a todas las personas que me han acompañado en este proceso, amigos, compañeros de trabajo, ex compañeros de trabajo.

Muchas gracias a todos.

CONTENIDO

INTRODUCCIÓN

Llamado como el sucesor de OpenGL el día de su anunciamiento, Vulkan es una API de gráficos 3D moderna, diseñada para hacer uso de todo el poder computacional de la plataforma en la que se encuentra corriendo, contando con mejoras en su arquitectura las cuales permiten mayor desempeño a costo de mayor tiempo de desarrollo y curva de aprendizaje.

OpenGL desde el día de su nacimiento ha pasado por 19 versiones (De la 1.0 en 1992 a la 4.6 en 2017), en las cuales ha sufrido muchos cambios en su arquitectura y su API ha tenido bastante obsolescencia para acomodarse a las necesidades de las tecnologías actuales como lo que es GPGPU, soporte para programación por Shaders y multithreading.

Una de las decisiones de diseño de OpenGL es ocultar toda la interacción con el driver del programador, lo cual a lo largo del tiempo ha generado una sobrecarga en las aplicaciones desarrolladas con este, produciendo disminuciones de rendimiento e incluso afectando el uso de energía del equipo en el cual se ejecuta.

Para tratar las desventajas que OpenGL tiene con las necesidades del mercado actual, Vulkan fué creado, adoptando varias filosofías de diseño pensadas para solucionarlas (Olson, 2016), algunas de ellas son:

- Dar Control Explícito, nada de “magia” sucede en el driver.
- Multithreading, todos los objetos son visibles y accesibles desde cualquier thread.
- Desempeño más predecible.
- Soporte para dispositivos móviles.

El desarrollo de este proyecto tiene como objetivo principal realizar una comparativa de desempeño entre OpenGL y Vulkan, buscando darle un punto de partida a los desarrolladores para elegir qué API usar en sus aplicaciones. Así mismo todo el estudio se basó en 2 hipótesis, la primera, que Vulkan al disminuir la carga del Driver mejora el desempeño en la CPU, la segunda, que al disminuir la sobrecarga en la GPU, el consumo de energía se ve disminuido en Vulkan. En los resultados experimentales se ve reflejado que ambas hipótesis pudieron ser comprobadas mostrando que Vulkan tiene mejor uso de CPU y optimiza el consumo de energía en la GPU.

ESTADO DEL ARTE

HISTORIA

NACIMIENTO. La historia del nacimiento de OpenGL está directamente conectada con el surgimiento de Silicon Graphics. Durante los años 1970 y 1980 la mayoría de videojuegos corrían sobre sistemas especializados, no existía la posibilidad de ver contenidos digitales, tales como películas o animaciones. Fue en 1981 cuando apareció Silicon Graphics, una compañía especializada en el gráficos 3D, que desarrollaba software y hardware para este objetivo, uno de estos software fue el sistema operativo IRIX, el cual fue uno de los primeros sistemas operativos con una interfaz de usuario la cual proveía una herramienta que le facilitaba a los desarrolladores realizar gráficos por computador llamada IRIS GL.

IRIS GL era una herramienta muy avanzada para la época, hizo más fácil el desarrollo de aplicaciones gráficas. Pero aún seguía existiendo un problema, no había un estándar para poder realizar desarrollo en otras plataformas puesto que IRIS GL solo corría en máquinas de trabajo desarrolladas por Silicon Graphics, fuera de eso que seguía siendo una herramienta privativa.

Fue así como en 1992 por decisión de Silicon Graphics, realizaron una limpieza de la API de IRIS GL y la lanzaron al público con otro nombre, OpenGL (Open Graphics Library).

TRAYECTORIA Desde su nacimiento, OpenGL ha sido una de las API de gráficos 3D más usadas junto a su competencia DirectX desarrollada por Microsoft, han pasado ya más de dos décadas desde esto y OpenGL ha cambiado mucho desde su nacimiento, con el surgimiento de hardware más especializado y la necesidad de flexibilidad requerida por parte de los implementadores.

La historia de Vulkan empieza con Mantle una API desarrollada por AMD la que posteriormente en el 2015 fue acogida por Khronos Group, un consorcio formado por varias empresas involucradas en la industria multimedia, y la liberaron al público con el nombre de Vulkan.

Vulkan surgió con el propósito de ser una API de gráficos 3D que cumplía con las expectativas de la tecnología actual, por esta razón buscaron solucionar los mayores inconvenientes que presentaba OpenGL, tales como tener cuellos de botella en CPU, limitar el manejo directo de la GPU y el manejo de errores de la API.

A partir de esto salió una nueva API con el eslogan “Industry Forged” ("Vulkan - Industry Forged", 2019) el cual fue dado al ser apoyado por muchas compañías grandes de la industria ("Khronos Members", 2019), dando el paso a un software estandarizado.

Vulkan sigue creciendo, teniendo un desarrollo activo desde su anuncio en 2015 y versiones nuevas entregadas mensualmente (“Vulkan Docs Releases”, 2019) adicionalmente a esto se han desarrollado por parte de sus colaboradores muchas extensiones las cuales dan soporte a funcionalidades específicas de cierto Hardware.

ESTUDIOS REALIZADOS

Titulo	Evaluating the Performance and Energy Efficiency of OpenGL and Vulkan on a Graphics Rendering Server
Electronic ISBN	978-1-5386-9223-3
Fecha	11 April 2019
Resumen	Con el objetivo de estudiar el desempeño y el gasto de energía de Vulkan y OpenGL ES en un servidor de renderizado, este documento publicado en la Conferencia internacional sobre informática, redes y comunicaciones (ICNC), realiza unas pruebas sobre estas dos tecnologías llegando como conclusión que Vulkan supera a OpenGL ES, gastando un 50% menos de energía, sin afectar el desempeño de la aplicación, dando a entender que si el consumo de poder es una preocupación Vulkan sería la mejor alternativa.
Aporte a la tesis	Gran de las pruebas y estudios realizados en este documento pueden ser puestos en práctica en el desarrollo de esta monografía, con la gran diferencia de que: 1. Se hará uso de OpenGL en vez de OpenGL ES 2. La maquía sobre las que correrán las pruebas es un computador personal el cual cuenta con menos capacidad computacional.

Titulo	VComputeBench: A Vulkan Benchmark Suite for GPGPU on Mobile and Embedded GPUs
Electronic ISBN	978-1-5386-6780-4
Fecha	13 December 2018
Resumen	En recientes años se ha hecho muy atractivo el uso de las GPGPU (General Purpose GPU) para diferentes tareas computacionales de alto costo, para esto actualmente en el mercado existen algunas tecnologías las cuales hacen uso de GPUs modernas, entre ellas las más conocidas son CUDA y OpenGL. Este documento tiene el propósito de mostrar Vulkan como una alternativa para programación en GPGPUs, exponiendo una herramienta llamada VComputeBench la cual realiza pruebas de desempeño de diferentes algoritmos sobre la interfaz de Vulkan para realizar procesamiento computacional; al probarlo sobre resultados de los mismos algoritmos ejecutados sobre CUDA y OpenGL se concluye que aunque en algunos casos se ve mejoras en tiempo del 53% y 66% ponen a discusión el gran esfuerzo en programación que conlleva.x
Aporte a la tesis	A pesar que esta investigación trata sobre el uso de Vulkan como un mecanismo de computación general, da un amplio conocimiento de la capacidad computacional de la API, de esta manera ver que resultados

	puede dar en la parte gráfica.

Titulo	Real-Time GPU-based SPH Fluid Simulation Using Vulkan and OpenGL Compute Shaders
Electronic ISBN	978-1-5386-5813-0
Fecha	12 November 2018
Resumen	Buscando comprobar el desempeño y potencial de Vulkan al realizar Smoothed Particle Hydrodynamics (SPH), un método computacional usado para realizar simulación de líquidos haciendo uso del poder computacional de la GPU, este documento realiza dos casos de pruebas, mostrando la parte matemática que usa estos modelos y un pseudo código el cual es implementado en GLSL y posteriormente ejecutados tanto en Vulkan como OpenGL, llegando a la conclusión que si el número de partículas usado es mayor a 30,000 Vulkan se desempeña más rápido, pero si es menor a 20,000 su desempeño es peor que OpenGL.
Aporte a la tesis	En este estudio realizado se puede mirar cómo impacta la cantidad de elementos renderizados en el desempeño que puede proveer una u otra tecnología lo cual sirve para fundamentar tests los cuales incluyan

	tanto rangos pequeños y altos de elementos renderizados.

En la actualidad OpenGL sigue siendo la principal alternativa a la hora de seleccionar una API de gráficos 3D, pero con el el avance tecnológico actual y la llegada de tecnologías de realidad virtual Vulkan ha tomado bastante auge.

Motores de videojuegos famosos tales como Unreal Engine 4 o Unity 5 ya al momento soportan esta nueva tecnología y muchos más cada vez se unen a esto, a su vez consolas de videojuegos tales como PlayStation 4 y Nintendo Switch también están soportándolo y muchas otras plataformas más lo hacen hasta el momento, inclusive en sistemas Apple, los cuales estaban restringidos a usar su propia API, ya hay una alternativa para poder usar Vulkan, llamada MoltenVK, así dándole a Vulkan una salida más grande al mercado de la industria de desarrollo de videojuegos.

MARCO TEÓRICO

ARQUITECTURA DE LAS SDK

COMPARATIVA. Siendo OpenGL una API de más de 2 décadas de antigüedad su diseño original ya no se adapta al hardware actual el cual es mucho más complejo al tener flujos más específicos, muchas de las tareas que deberían ser responsabilidad de la aplicación son hechas por el driver, dejando así poco control explícito de la arquitectura de la GPU, y forzando al desarrollador a agregar una sobrecarga posiblemente innecesaria a su programa.

Como podemos analizar en la siguiente figura, Vulkan tiene una ventaja con respecto a OpenGL y es su manejo explícito de la GPU del sistema

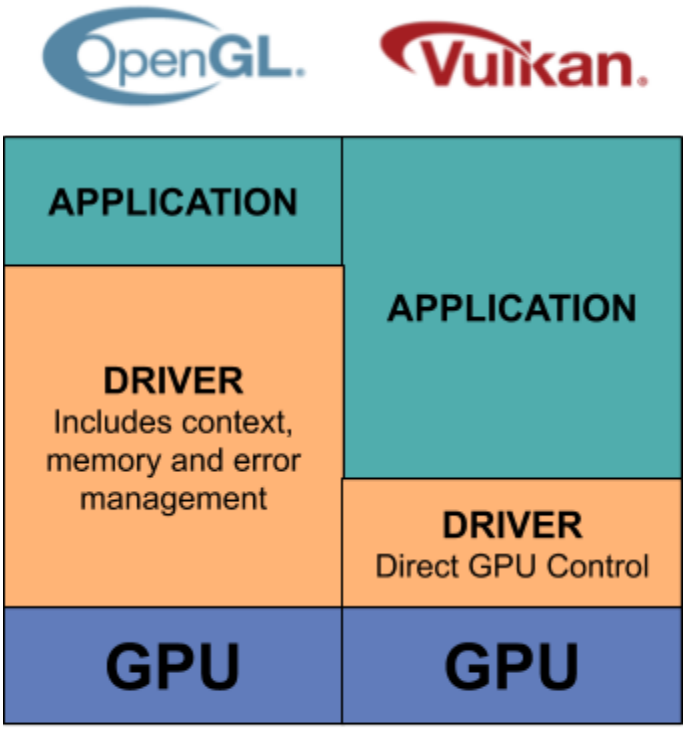

Figura 1: Arquitectura general de OpenGL y Vulkan

Mientras que OpenGL se encarga de sobrecargar el driver con operaciones realmente no necesarias en esta capa, tales como la creación de contexto, el manejo de errores y el manejo de memoria, Vulkan deja todo el trabajo del lado de la aplicación liberando gran carga en el driver y por consiguiente mejorando el desempeño en la CPU.

Al realizar una comparación de la tubería gráfica que usa OpenGL y Vulkan (Figura X y Y), se pudo observar que tienen una estructura base muy similar, su gran diferencia es en la capacidad configurable que tienen cada una de las etapas del flujo. En OpenGL cada una de las etapas fijas tiene un estado por defecto el cual no es configurable, Vulkan al contrario, cada una de las etapas tiene que ser configurada previamente por el usuario, dando mayor flexibilidad y permitiendo ciertas optimizaciones dependiendo de los recursos que se quieran o no usar.

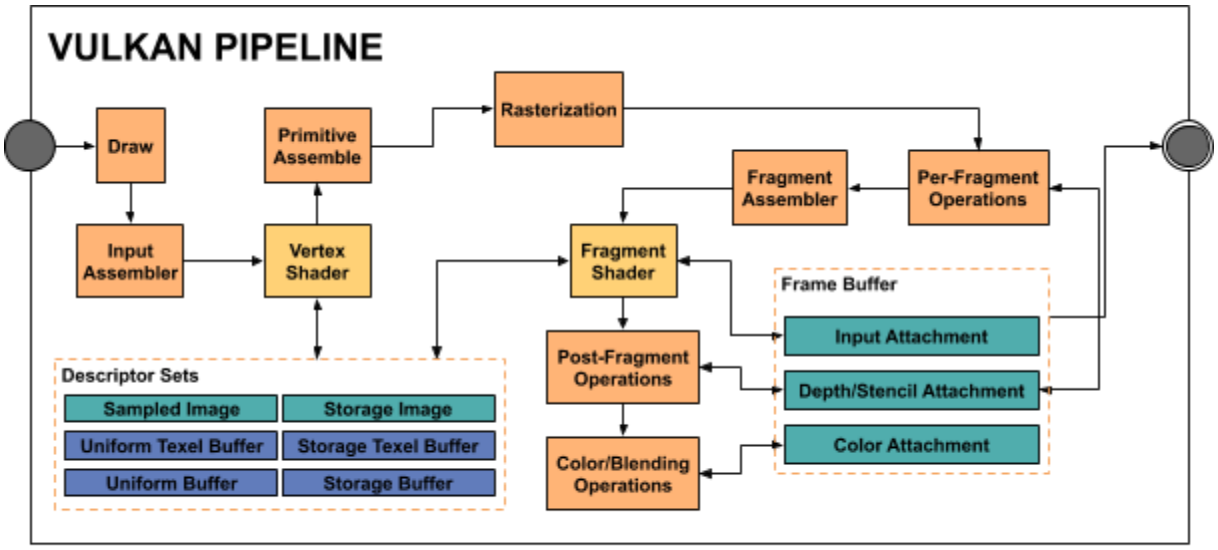

Figura 2: Tubería de renderizado base de Vulkan

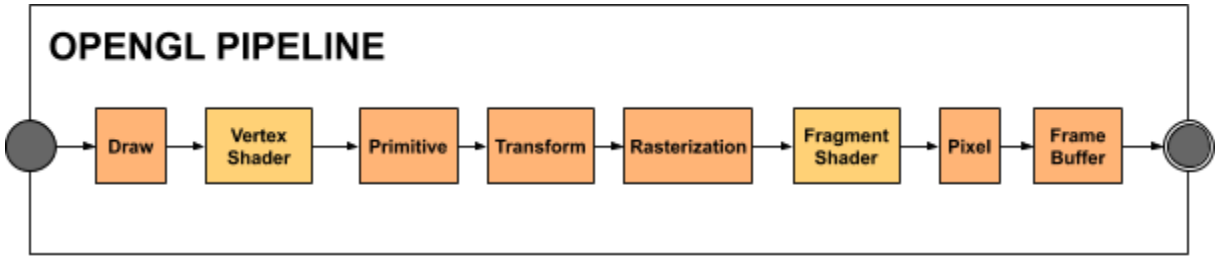

Figura 3: Tubería de renderizado base de OpenGL

Vulkan introduce un concepto nuevo a su tubería gráfica, llamado Descriptor Sets, estos son encargados de proveer recursos los cuales son utilizados a través de todos los componentes programables de la arquitectura, esta información puede ser desde datos, como vectores o matrices, o imágenes.

API

SWAP CHAIN. Para realizar la presentación es necesario que el la imagen resultado que está almacenada en el Frame Buffer sea mostrada en pantalla, teniendo en cuenta que una GPU corre a mayor frecuencia que la tasa de refresco de un monitor, que es regularmente de 60HZ. Es necesario encontrar una manera de estabilizar la velocidad de refresco de la aplicación para así evitar problemas como el stuttering, a este técnica se le conoce como Swap Chain y existen diferentes métodos que se pueden aplicar para esto uno de los más famosos es el double buffering el cual consiste en que mientras un Frame Buffer es mostrado en pantalla, otro está siendo usado en el proceso de renderizado; como este hay muchas más técnicas.

Para este proceso Vulkan maneja varios modos de presentación los cuales dependiendo del dispositivo físico tienen soporte o no, en contraparte OpenGL no tiene esta funcionalidad

incluida en su API y el funcionamiento de esto depende propiamente del driver que está usando la máquina.

MODELO DE COMUNICACIÓN. Realizando un análisis de la manera en que Vulkan y OpenGL realizan el envío de información desde su API a la GPU se pudo ver que el modelo cambia totalmente en Vulkan adoptando un modelo de encolamiento por medio de Command Buffers, estructuras de datos utilizadas para realizar el envío de comandos los cuales le especifican al driver que acción debe hacer y en qué sección de la tubería debe ejecutar esto. Estos Command Buffers son ejecutados por medio de Colas las cuales tienen una funcionalidad específica, para gráficos, computación y presentación, y son seleccionadas dependiendo de los comandos que se quieran ejecutar y el dispositivo físico que se esté usando.

La comunicación usada por OpenGL es manejada directamente por el driver, y se accede por medio de métodos de la API los cuales reciben directamente la información que se va a procesar. Estas líneas de comunicación son creadas por defecto sin ningún control en su cantidad o uso, lo cual genera que gran sobrecarga en el driver.

RUTINAS DE SINCRONIZACIÓN. Estos métodos usados por las API 3D cumplen la función de sincronizar las operaciones realizadas por la CPU y la GPU para que su funcionamiento sea correcto en el flujo del pipeline.

Vulkan realiza este proceso utilizando varias estructuras de sincronización entre ellas Semáforos, Barreras de Memoria (Fences), Eventos, entre otros, estas rutinas funcionan controlando el acceso a recursos que hace uso esta API, todos estos accesos tiene que ser explícitamente

especificados a la hora de su implementación, lo que permite acceder a ciertas optimizaciones al ubicar los recursos directamente en la etapa de la tubería que deben funcionar.

OpenGL realiza este proceso implícitamente, dejando todo el trabajo de parte del driver lo cual puede generar demoras en la sincronización de ciertos recursos al no saber en qué etapa de la tubería deben consumirse.

SHADERS. Una de las herramientas más importantes en la computación gráfica moderna es la capacidad que tienen los softwares de poder correr instrucciones dentro de la GPU, los shaders son estos programas los cuales cumplen cierta función específica (Manejo de iluminación, color, transformaciones, creación de geometría, entre otros) y corren dentro de una etapa específica del pipeline anteriormente mencionado cada API gráfica hace uso de algún lenguaje en especial para poder escribir estos programas, en el caso de OpenGL el lenguaje usado es GLSL (OpenGL Shading Language), Vulkan a su vez hace uso de un lenguaje en formato de bits no legible llamado SPIR-V.

Para facilitar su uso y escritura, shaders creados en GLSL pueden ser transformados a SPIR-V teniendo en cuenta algunas restricciones dadas por la especificación, y para esto existen algunas herramientas como lo son glslang o shaderc las cuales compilan GLSL transformando el código en el formato de bits, SPIR-V.

Algunas ventajas que se pudieron encontrar a la hora de ejecutar las pruebas es que al SPIR-V ser un lenguaje que es muy cercano al lenguaje de máquina el computador no necesita realizar una proceso de interpretación tan arduo, en cambio, shaders escritos y cargados directamente sobre GLSL tienen que pasar por un proceso de compilación en tiempo de ejecución lo cual

genera tiempo adicional en la inicialización, volviéndose un tiempo bastante considerable cuando se tiene un software que contiene muchos de estos.

MANEJO DE ERRORES. Una API bien arquitecturada tiene que tener un soporte robusto para verificar el correcto funcionamiento de la misma, esto considerando que como seres humanos estamos siempre propensos a cometer ERRORES, por esta razón OpenGL y Vulkan no son la excepción, aunque ambas utilizan métodos distintos para hacer esto, su objetivo es el mismo, verificar que usemos cada una correctamente.

OpenGL al llevar más tiempo en el mercado, adopta un sistema de manejo de errores antiguo y que en épocas modernas se puede considerar obsoleto. Su problema principal radica en el hecho que para poder realizar un manejo de errores correcto es necesario estar llamando una función específica de la API la cual nos notifica con una cadena de texto si existió o no un error en la última instrucción realizada, esta función se llama glGetError. Realizar esto a largo plazo hace que un programa se vuelva largo y poco legible.

Por otro lado Vulkan adapta un concepto nuevo poco utilizado en APIs actuales, llamado capas de validación (Validation Layers por su nombre en inglés), estas son encargadas de inyectar código el cual se ejecuta en cada llamado a la API, su mayor ventaja es al ser código inyectado se puede optar por no usarlo, lo cual removerá toda la sobrecarga que estas generan.

METODOLOGÍA

Para el desarrollo del problema se hicieron varias pruebas cada una con diferente cantidad de modelos renderizados en pantalla: 50, 100, 200, 400 y 600; para esto se hizo uso del modelo de prueba del conejo de Stanford (Turk, 2000), el cual consiste en 69,451 triángulos y ha sido usado para la prueba de diferentes algoritmos gráficos.

Las pruebas fueron realizadas sobre un computador con las siguientes características:

- Procesador (CPU): AMD Ryzen 5 2600X
- RAM: 32GiB DDR4 3000MHz
- Tarjeta de Video (GPU): Nvidia GTX 1060 6GB

Cada prueba fue ejecutada por un total de 5 minutos tanto para OpenGL como para Vulkan, en los cuales cada segundo se realizaba la captura de datos del computador tales como uso de CPU, uso de GPU, consumo de memoria RAM, consumo de memoria VRAM, y consumo energía por parte de la CPU. Para la medición de datos de la GPU se hizo uso de la librería NVML ("NVIDIA Management Library", 2017) que provee Nvidia y los datos de RAM y CPU fueron capturados del PSAPI ("About PSAPI", 2018) y Performance Counters API ("Performance Counters", 2018), dos API que provee Windows para obtener información relevante del proceso actual.

El desarrollo de esta tesis se basa en dos hipótesis, la primera es que Vulkan hace mejor uso de la CPU liberando mucha carga del driver y por consiguiente mejorando su porcentaje de uso, la segunda fué basada en una prueba hecha por ARM ("Vulkan API vs OpenGL ES in Unity: Get

10-12% extra playtime on mobile!", 2017) en la cual muestran que en Vulkan el uso de energía es mucho menor comparado con OpenGL ES, en este caso se quería comprobar lo mismo con OpenGL.

La comparación de los resultados de cada una de las pruebas se puede encontrar en los siguientes gráficos:

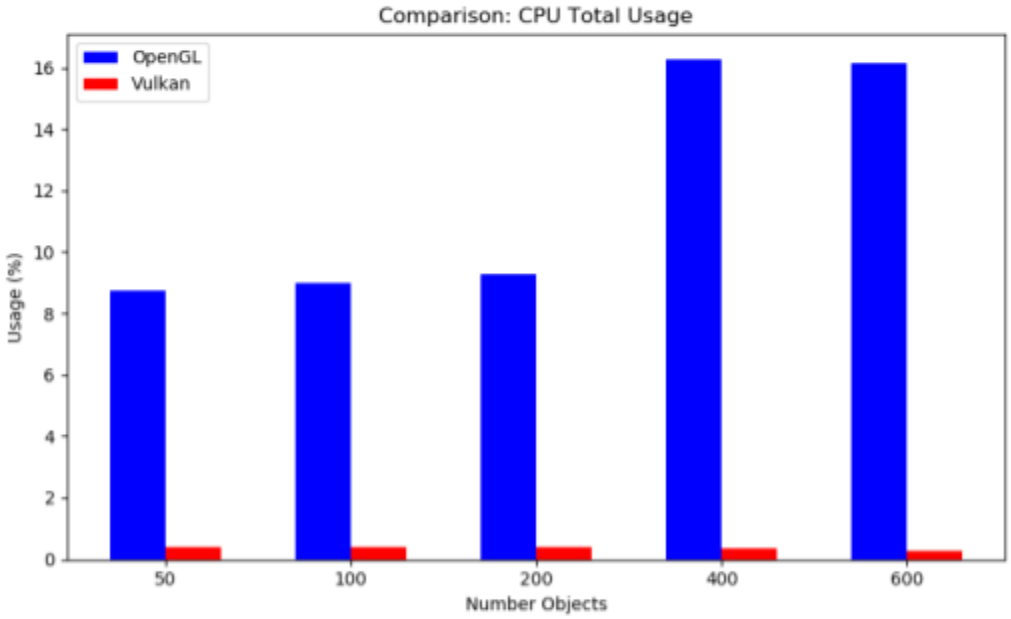

Figura 4: Uso total de CPU de Vulkan y OpenGL

# of Objects	VK Usage (%)	GL Usage (%)	Performance Gain (%)
600	0.27196	16.15648	98.31674
400	0.34559	16.28594	97.87801
200	0.41401	9.27728	95.53737
100	0.40626	8.99084	95.48135
50	0.41495	8.72752	95.24546
	0.37055	11.88761	**96.49178**

Figura 5: Tabla comparativa de uso de CPU

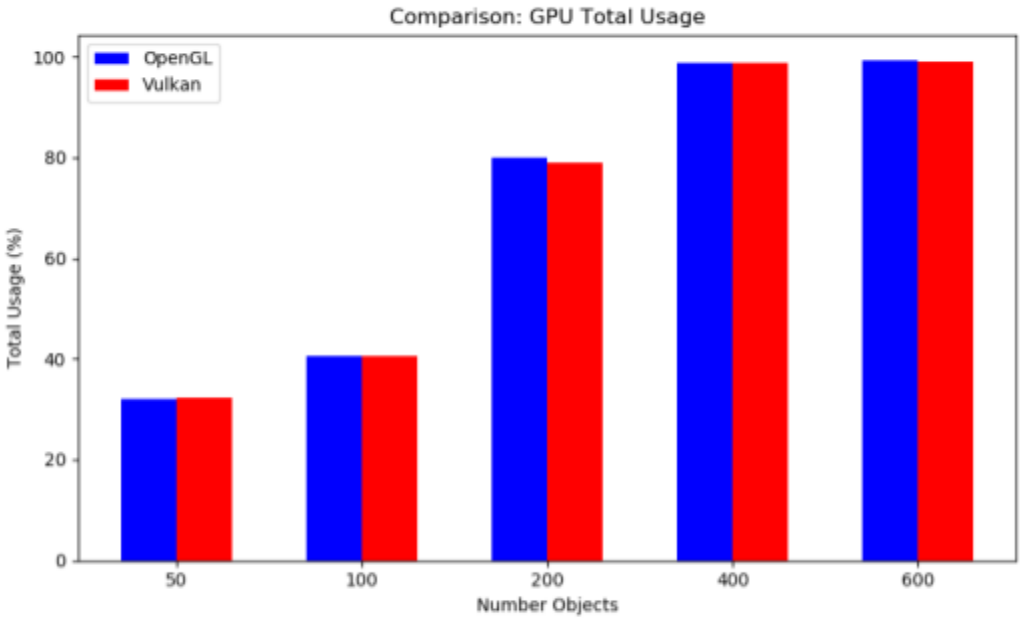

Figura 6: Uso total de GPU de Vulkan y OpenGL

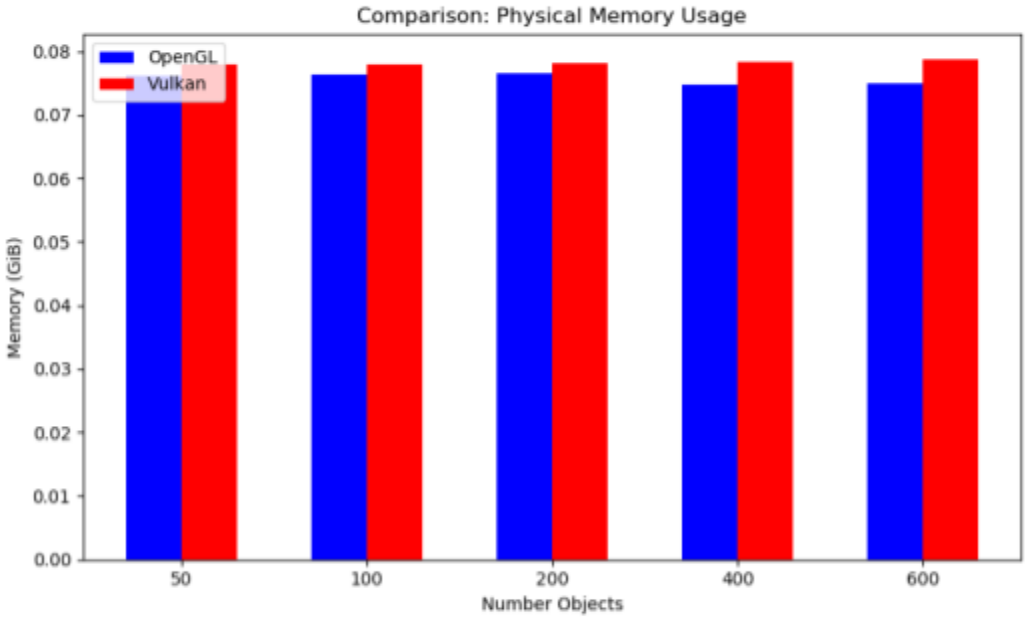

Figura 7: Uso total de RAM de Vulkan y OpenGL

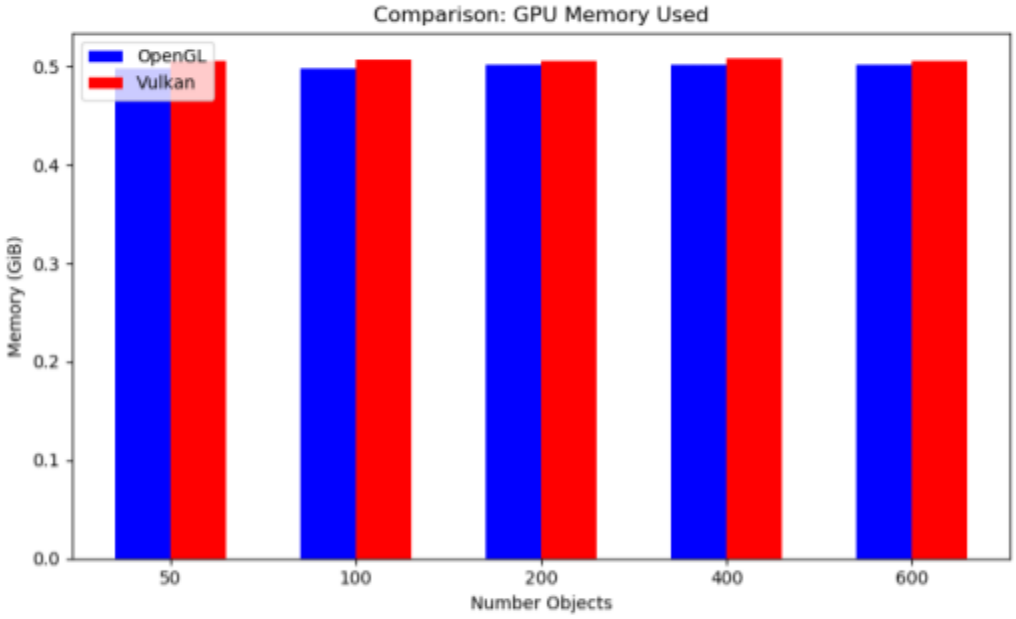

Figura 8: Uso total de VRAM de Vulkan y OpenGL

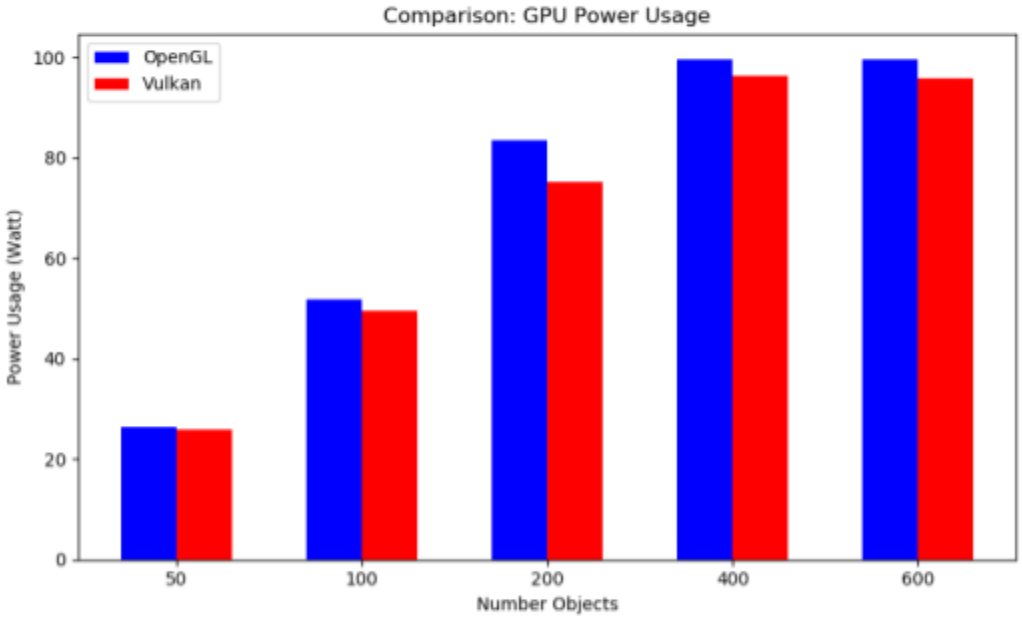

Figura 9: Consumo de energía de Vulkan y OpenGL

# of Objects	VK Power Usage (W)	GL Power Usage (W)	Energy Saving (%)
600	95845.84667	99721.27333	3.88626
400	96339.39333	99534.21667	3.20977
200	75167.17000	83588.92333	10.07520
100	49469.06333	51667.61333	4.25518
50	25952.62333	26333.94000	1.44800
	68554.81933	72169.19333	**4.57488**

Figura 10: Tabla comparativa de uso de energía

Con los resultados obtenidos se pudo comprobar las hipótesis, donde se encontró que Vulkan tiene una mejora del uso de CPU de un 96.5% comparado con OpenGL y se encontraron mejoras

con respecto al uso de energía en un 4.6% al hacer uso de Vulkan más cuando la cantidad de objetos en pantalla era muy alta.

ANÁLISIS Y CONCLUSIONES

En el desarrollo de este documento se realizó una comparación de desempeño entre dos tecnologías de renderizado 3D, Vulkan y OpenGL, donde se pudo comprobar que para las pruebas experimentales realizadas el desempeño Vulkan excede el alcanzado por OpenGL cuando hablamos de uso de CPU y menor consumo de energía en la GPU. Si a la hora de realizar algún software es importante alguna de estas características, por ejemplo: se está desarrollando un software para dispositivos embebidos o consolas de videojuegos que tienen limitaciones en cuanto al hardware, se puede optar por Vulkan como la mejor opción, eso sí, teniendo en cuenta que el tiempo de desarrollo y la curva de aprendizaje de dicho software es alto, y muchos desarrolladores no estarían dispuestos a afrontarlos.

Finalmente se concluye:

1. Se encontró que Vulkan tiene un mejor desempeño y uso de CPU comparado con OpenGL.
2. El consumo de energía fue menor en las pruebas realizadas en Vulkan.
3. A pesar de las ventajas que claramente Vulkan provee versus OpenGL, una de sus desventajas claras es la curva de aprendizaje, a pesar de ser una API más nueva, al dar control explícito es necesario aprender muchos conceptos para entender cómo funciona el flujo en la API lo que incrementa considerablemente la curva de aprendizaje.

BIBLIOGRAFÍA

- Olson, T. (2016). Vulkan 101. Retrieved from https://www.khronos.org/assets/uploads/developers/library/2016-vulkan-devday-uk/1-Vulkan_101.pdf
- Vulkan - Industry Forged. (2019). Retrieved August 2019, from https://www.khronos.org/vulkan
- Khronos Members. (2019) . Retrieved August 2019, from https://www.khronos.org/members/list
- Vulkan Docs Releases. (2019). Retrieved October 2019, from https://github.com/KhronosGroup/Vulkan-Docs/releases
- Turk, G. (2000). The Stanford Bunny. Retrieved from http://www.cc.gatech.edu/~turk/bunny/bunny.html
- NVIDIA Management Library. (2017). Retrieved from https://developer.nvidia.com/nvidia-management-library-nvml
- About PSAPI. (2018). Retrieved from https://docs.microsoft.com/en-us/windows/win32/psapi/about-psapi
- Performance Counters. (2018). Retrieved from https://docs.microsoft.com/en-us/windows/win32/perfctrs/performance-counters-portal
- Vulkan API vs OpenGL ES in Unity: Get 10-12% extra playtime on mobile!. (2017). Retrieved from http://www.youtube.com/watch?v=WI7nXq8oozw